C++学习笔记总结4，5，6，7

2022年3月19日 21:00

- 类
  - 类的定义
    - 简单来说，类就是C++对象模式的具体体现
    - 对象的三大特征：封装，多态，继承
    - 类的三大分类：public，private，protect（不常用）
      - public：对外公开的，全域的代码都可以访问
      - private：不对外公开的（封装的）
  - 类的分类
    - struct
      - 对于struct来说，如果他是类，则类内的所有成员都是公共成员
    - class
      - 对于class来说，分为public和private，默认状态下（即不分类public和private）只有public
  - private
    - 在private下面的变量/函数，都是被分装起来的，不能直接被访问
    - 如何访问
      - 在友元函数内部
      - 在public作用域的函数内部
    - 调用
      - 我们可以使用this指针（this->xx）在函数框架内部显式的调用成员，也可以直接调用成员（隐式的调用）
    - 初始化
      - 可以将类内的变量赋予一个值，进行初始化，保证类内的值不出差错
    - 编写函数
      - 在函数的内部编写函数的时候默认式inline函数，从而不能有太长的字数
      - 在函数的外部编写函数的时候需要加上作用域 class：：xx
      - 在函数内部可以直接修改成员的值
      - 遵循普通函数的原则，可以重载
  - public
    - 在public下面的变量/函数，是可以直接访问的
    - 调用
      - 对于一个已经创建好的类，我们使用点运算符直接对共有成员进行调用
      - 如果在调用的时候对函数 的结尾声明为const，则无法使用指针（this）显式或者隐式的更改值
    - 构造函数
      - 构造函数没有const

        - 构造函数的形式
            - xxx（）=default（默认构造函数）
            - xxx（string a，string b）：member1（a），member2（b）{};
            - 委托构造函数xxx（string a）：xxx（a，0，0）；
        - 构造函数需要按照顺序进行初始化操作
    - 友元函数
        - 以一条friend声明放在class的最顶端，表示这个函数可以访问作用域内的私有成员
        - 与public函数内部直接访问不同，他需要靠点运算符才能访问
        - 在声明完class后在正式的函数开头是需要再次声明的
    - 注意
        - 注意函数命名时候的返回类型，如果返回class类型就可能可以多重调用
- IO
    - iostream
        - 作为文件内部的io类，用于支持输入和输出
        - iostream类的变量
            - cin：作为不用定义，添加即有的输入类，可以直接向cin中输入你想要的内容，后经过流运算符>>将其输入对应的变量
            - cout：作为不用定义，添加即有的输出类
                - 可以使用io迭代器直接向cout中写入需要的东西
                - 可以使用流运算符<<把变量的物品写入cout，进行输出
    - fstream
        - 作为对文件进行写入的io类，用于支持文件的输出和输出
        - fstream变量
            - ifstream
                - ifstream可以写自己的名字例如：ifstream in，in就充当了和cin一样的作用
                - 与istream相同，ifstream也是可以按照顺序依次读取文本中的内容，默认忽略第一个空格
            - ofstream
                - ofstream可以写自己的名字例如：ofstream out，out就充当了和cout一样的作用
        - fstream的函数
            - 当我们想利用fstream打开/关闭文件的时候有下面两种方式
                - fstream.open（） 显式的打开文件
                - fstream xx(文件路径) 隐式的打开文件
                - xx。close（） 关闭文件
            - 当文件被打开后，我们可以使用文件模式对其加以调整
                - ifstream in（文件，ifstream：：app）
                - 常见的文件模式有
                    - in：以读的方式打开文件

        - ◇ out：以写的方式打开文件
        - ◇ app：每次打开文件就立刻定位到文件末尾
        - ◇ ate：打开文件后立刻定位到文件末尾
        - ◇ binary：以二进制的方式打开文件
    - sstream
        - 作为一个对文件进行写入和输出的IO类，用于对一个字符串进行输入和输出
        - 表 8.5：stringstream 特有的操作

sstream strm;	strm 是一个未绑定的 stringstream 对象。*sstream* 是头文件 sstream 中定义的一个类型
sstream strm(s);	strm 是一个 *sstream* 对象，保存 string s 的一个拷贝。此构造函数是 explicit 的（参见 7.5.4 节，第 265 页）
strm.str()	返回 strm 所保存的 string 的拷贝
strm.str(s)	将 string s 拷贝到 strm 中。返回 void

    - io流的调整
        - io流的写入方式
            - □ 所有的io流都是通过定义的”变量“发挥作用的的
              ifstream in 所有东西写入到in，利用》可以写入到其他的变量去
            - □ 可以利用io的迭代器进行变量的写入
              istream-iterator<int> a(cin) a是一个迭代器，指向cin
        - io流的管理
            - □ io流有两种损坏的状态，bad和fail，我们可以使用xxx.clear（）来复位流
            - □ 流的刷新
                - ◆ 流的刷新三个条件，满足一个即可使用：程序结束，endl，缓冲区慢
                    - ◇ 程序出错可能不会立刻刷新流，会影响到下面的操作
                - ◆ 立刻刷新流：unibuf取消立刻刷新流：nounibuf
            - □ 当一个大括号进行结束的时候会默认销毁流，如果在大括号内重复使用需要刷新流（cin。clear（））
- 顺序容器
    - 顺序容器分类为连续容器（指内存空间上连续分布的容器）和不连续的容器
    - 顺序容器通用格式
        - 每个顺序容器都有名称+变量构成一个大的变量名字+新的变量名字
          vector<int> a
    - 顺序容器的通用方法
        - 初始化
            - □ 使用花括号直接指定初始化一部分元素
              vector<int> a{1,2,3,4,5,6}
            - □ 使用圆括号批量指定初始化一部分元素
              vector<int> a(10,2); 有10个元素每个都是2
    - 连续容器
        - 操作
            - □ 直接操作顺序容器

表 9.5：向顺序容器添加元素的操作

这些操作会改变容器的大小；array 不支持这些操作。
forward_list 有自己专有版本的 insert 和 emplace；参见 9.3.4 节（第 312 页）。
forward_list 不支持 push_back 和 emplace_back。
vector 和 string 不支持 push_front 和 emplace_front。

c.push_back(t) c.emplace_back(*args*)	在 c 的尾部创建一个值为 t 或由 *args* 创建的元素。返回 void
c.push_front(t) c.emplace_front(*args*)	在 c 的头部创建一个值为 t 或由 *args* 创建的元素。返回 void
c.insert(p,t) c.emplace(p,*args*)	在迭代器 p 指向的元素之前创建一个值为 t 或由 *args* 创建的元素。返回指向新添加的元素的迭代器
c.insert(p,n,t)	在迭代器 p 指向的元素之前插入 n 个值为 t 的元素。返回指向新添加的第一个元素的迭代器；若 n 为 0，则返回 p
c.insert(p,b,e)	将迭代器 b 和 e 指定的范围内的元素插入到迭代器 p 指向的元素之前。b 和 e 不能指向 c 中的元素。返回指向新添加的第一个元素的迭代器；若范围为空，则返回 p
c.insert(p,il)	il 是一个花括号包围的元素值列表。将这些给定值插入到迭代器 p 指向的元素之前。返回指向新添加的第一个元素的迭代器；若列表为空，则返回 p

WARNING 向一个 vector、string 或 deque 插入元素会使所有指向容器的迭代器、引用和指针失效。

- □ 迭代器操作顺序容器
  - ◆ 见迭代器----顺序容器

▪ 连续容器里面分为可以动态扩充大小的容器和不可以动态扩充大小的容器

- □ 动态扩充大小的容器
  - ◆ vector
    - ◇ 操作：类似链表，可以不断的扩充容器，但是空间上又是连续的，本质上是一个不断扩容的数组
    - ◇ vector不支持有front字眼的操作
  - ◆ string
    - ◇ 支持下标操作
    - ◇ 支持下标当作迭代器传入：&a【2】
  - ◆ deque（队列）
    - ◇ 同上，但是会慢
    - ◇ 在头头和尾部操作是最快的
  - ◆ stack（队列）
    - ◇ 支持下标，支持迭代器
    - ◇ 在一头操作最快
- □ 不可以动态扩充大小的容器
  - ◆ array
    - ◇ 支持下标操作
    - ◇ 与上述容器不同的是容器不可以扩容，容器的容量是固定的
    - ◇ array在删除的时候，不会删除元素
    - ◇ array在创建的时候，同时指定值的类型和值的大小
      array<int,10> a;

○ 不连续容器

▪ 不连续容器有着自己特殊的算法和操作，他都直接定义在了对于的顺序容器里面

▪ forward-list

        - 单向链表，不支持随机访问，只是支持++
    - list
        - 双向访问，支持--但是并不支持随机访问
    - 不连续容器在删除的时候，会真正的删除元素（unique函数等等）
- 迭代器
    - 迭代器的分类
        - 输入迭代器：只读不写，单遍扫描，只能递增
            - 支持运算符
                - 相等或不相等运算符==/！=
                - 递增运算符++
                - 读取元素，解引用运算符*
                - 箭头运算符-》/（*xx）。xxx
            - 输入迭代器只用于顺序访问，对于一个输入迭代器，*it++保证是有效的，但递增他可能导致其他所有指向流的迭代器失效
            - 常见的输入迭代器：istream-iterator
        - 输出迭代器：只写不读，单边扫描，只能递增
            - 支持运算符
                - 递增运算符++
                - 解引用运算符（*）
            - 我们只能向一个输出迭代器赋值一次，类似输入迭代器，输出迭代器只能用于单遍扫描算法
            - 常见：ostream-iterator
        - 前向迭代器：可读写，多遍扫描，只能递增
            - 支持
                - 所有输入输出迭代器的操作
                - 多次读写一个元素
                - 只能沿着一个方向移动
            - 常见：replace，forward-list
        - 双向迭代器：可读写，多遍扫描，可递增递减
            - 支持
                - 可以正向/反向读写序列中的元素
                - 支持所有前向迭代器的操作
                - 支持递减运算符
            - 常见：除了forward-list其他的都支持
        - 随机访问迭代器：可读写，多遍扫描，支持全部迭代器运算：全能
            - 常见：vector，array，deque，string，访问内置数组的指针
        - io类的迭代器（iterator定义）
            - istream-iterator，ostream-iterator属于io类迭代器
            - 方法：istream-iterator a（cin）---》a为cin的迭代器
            - cin可以替换为ifstream，istringstream的对于变量都可

        - 插入迭代器（iterator定义）
            - back—inserter：创建一个使用push-back的迭代器
            - front—inserter：创建一个使用push-front的迭代器
            - inserter：创建一个使用insert的迭代器，此函数接受第二个参数，这个参数必须是一个指向给定容器的迭代器，元素将被插入到给定迭代器所表示的元素之前
        - 反向迭代器（iterator定义）
            - rbegin，rend为代表的反向迭代器可以进行使用
            - 在使用的时候需要注意他们只支持有操作减号定义的容器
            - 不支持forward-list
        - 移动迭代器：用于移动迭代器（iterator定义）
    - 迭代器----顺序容器
        - 读取操作
            - *iter读取元素
        - 指向操作
            - iter类似于指针
        - 运算操作
            - 除了forward-list，其他的容器都支持++，--操作
        - 直接对容器操作的时候可能会让迭代器失效
        - 当我们使用添加和删除元素的容器操作时，必须注意这些操作可能使指向容器中元素的迭代器，指针或引用失效，很多会使迭代器失效的操作，如insert和erase，都会返回一个新的迭代器，，来帮助程序员维护容器中的位置。如果循环程序中改变了容器的大小的操作就要尤其小心其中迭代器，指针和引用的使用
- 算法
    - 算法是建立在迭代器之上的，算法并不改变容器
    - 算法的使用
        - 我们一般使用算法的时候，都是向里面传入
            - 两个迭代器，部分算法会返回一个迭代器
            - 三个迭代器，主要是copy函数使用
            - 两个迭代器，一个函数，主要是-if和compare的函数使用
            - 四个迭代器：范围复制
        - 少数算法可以传入一个函数（一个或者两个参数的，一般与函数有关）/谓词
    - lambda表达式
        - 传入函数的时候同时需要利用函数内部的值的时候使用
        - lambda必须使用尾置返回类型，可以把它看作一个函数单元进行操作

          【】（）mutible《可选，当要改变【】里面的值的时候使用》->{return；};
            - 【】
                - 用来抓取函数里面需要的变量
                    - 值抓取：当需要使用函数内部的值的时候，可以将函数内部的值拷贝一个到lambda来进行运算

        - 引用抓取：对于一些不能拷贝的个体（如ostream）建议用引用进行拷贝
        - 隐式抓取
            - =：利用值抓取的方式智能抓取变量中的东西
            - &：利用引用抓取的方式智能抓取变量中的东西
    - 不可以被省略
- ()
    - 里面可以装一到两个变量
    - 不可以被省略
- mutible：
    - 当要改变【】中抓取的值的时候可以使用
    - 可以省略
- ->
    - 指明返回类型
    - 默认情况下，当花括号里面有多个语句的时候，默认返回类型为void，此时如果需要返回则需要指定返回类型
    - ->int：代表返回一个int类型的变量
    - 可以省略
- {}
    - 函数体
    - 不可以省略
- 注意：
    - lambda是在创建的时候拷贝的变量，后续再对变量修改的时候不会改变拷贝变量的值

```cpp
#include <iostream>
using namespace std;
int main()
{
    int i=2;
    auto c=[i](){return i;};
    i++;
    int x=c();
    cout<<x<<endl;
}
```

```
问题  输出  终端  调试控制台

> Executing task: d:\codefield-study\CODE_C++\C++_SINGLE
\C++primer_exercise\bin\fanxing11.exe <

2

按任意键关闭终端。

build 任务
build 任务
run 任务
build 任务
run 任务
```

    - bind表达式
        - 需要include <functional>,using namespace placeholders;
        - bind函数返回一个函数，它可以将一个函数合并变量返回一个新的函数
          auto c=bind（xxx，a，-1，b）；
        - -1，-2作为占位符相当于lambda中（）放入的内容一样
        - 放入bind的表达式至少是x+1
        - bing与lambda的使用方式相同
- 注意
    - 。代表调用的操作，（）里面有着处理的对象